Reconfigurable Stream Processors for Wireless Base-stations

Sridhar Rajagopal* Scott Rixner Joseph R. Cavallaro
Rice University
{sridhar,rixner,cavallar}@rice.edu

Abstract

This paper presents the design and use of reconfigurable stream processors for the physical layer processing in wireless base-stations. Stream processors, traditionally used for high performance media processing, use clusters of functional units to provide support for hundreds of functional units in a programmable architecture. We provide hardware support for reconfiguration in stream processors, enabling them to be power-efficient by adapting to the compute requirements of the application. We demonstrate the real-time implementation of a 32-user wireless base-station, employing multiuser channel estimation, multiuser detection and Viterbi decoding physical layer algorithms, supporting a data rate of 128 Kbps/user. The reconfigurable stream processor runs at 1.2 GHz and has an estimated power consumption of 12.38 W at full workload. However, base-stations rarely operate at full capacity. When the base-station workload decreases, the reconfigurable stream processor adapts the number of clusters, functional units, voltage and frequency dynamically for power efficiency. When the application workload changes to 4 users, the reconfiguration support reduces the power to 300 mW at 433 MHz, providing a 41.27$\times$ decrease in power consumption. The cluster reconfiguration yields an additional 15-85% power savings over a stream processor with dynamic voltage and frequency scaling.

1. Introduction

Wireless base-stations require high computation rates, must perform those computations in real-time, and must limit their power consumption. In order to support multiple simultaneous users, a base-station must provide interference suppression. The complexity of interference suppression leads to a non-linear increase in computational demands as the number of users increases. Furthermore, power-efficiency is important, as it helps decrease operational and maintenance costs [1]. Increased power efficiency also allows smaller and lower cost battery backup systems, enabling continued, uninterrupted service during critical times of power failures such as the recent August blackout [2]. Computational capacity and power efficiency are becoming critical to base-station design as they become denser (pico-cells) and more prevalent in high-traffic areas.

As base-stations begin to provide higher data rates to individual users, flexibility becomes important in addition to computational capacity and power efficiency. Wireless systems are currently evolving from low data rate (Kbps) voice-only systems to high data rate (Mbps) multimedia systems, requiring billions of computations per second. These emerging wireless standards require greater flexibility than past standards, such as supporting different data rates, varying number of users, various decoding constraint lengths and rates, adaptive modulation and spreading schemes [3]. Programmability in the base-station provides this flexibility and makes the task of upgrading base-stations with emerging standards easier. As wireless standards evolve, programmable base-station designs will also allow limited versions of future standards to be implemented immediately without design time loss for new architectures.

Unfortunately, existing DSP architectures are not powerful enough for this new class of base-stations. Stream processors, however, have been shown to perform extremely well on media processing applications [4]. Programmable stream processors can efficiently support hundreds of arithmetic units, providing higher computational bandwidth in a power efficient manner. Previously, we have shown that because wireless baseband processing shares many similarities with media processing, stream processors are also good candidates for wireless base-stations [5]. Since real-time design constraints have to be met for the worst case, a full capacity base-station must employ enough resources to meet those constraints at a reasonable frequency and voltage. However, base-stations rarely operate at full capacity [6]. At lower capacity workloads, far fewer resources are required to meet the real-time constraints, so many of the resources will be used inefficiently.

We propose to dynamically adapt the resources of a stream processor to match the workload of the base-station, effectively turning an efficient media processor into an efficient wireless communications processor. Such a reconfigurable stream processor can adjust the frequency, voltage, and arithmetic resources, significantly reducing power dissipation under lightly loaded

*Contact information: 6100 Main St. MS-366, Rice University, Houston, TX 77005, Ph: 713-348-2256 Fax: 713-348-6196, sridhar@rice.edu

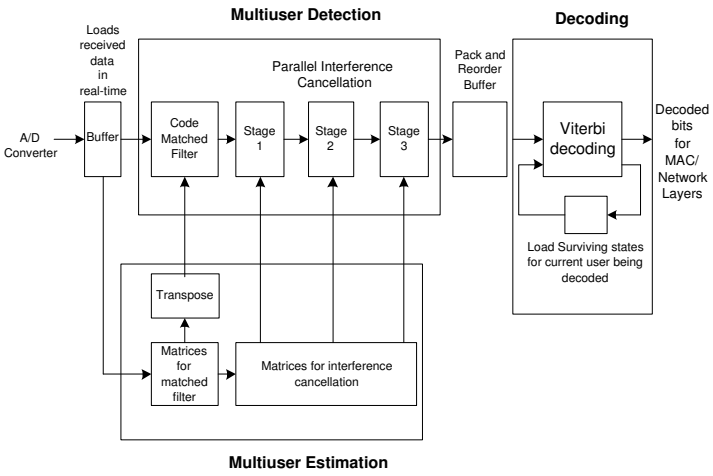

Figure 1. *Detailed view of physical layer processing at the base-station*

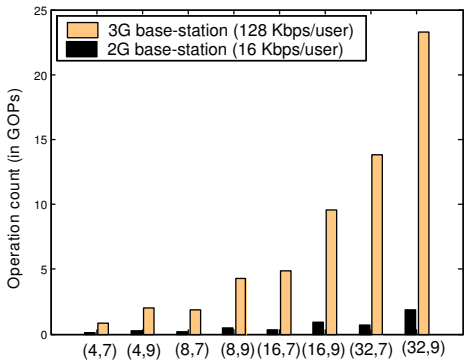

Figure 2. *Operation count for 2G and 3G base-stations (Users,Constraint lengths)*

conditions. For example, a full capacity base-station (32 users with strong Viterbi decoding) would operate the stream processor at 1.2 GHz and dissipate 12.4 W, whereas a lightly loaded base-station (4 users with weaker Viterbi decoding) would operate the stream processor at 433 MHz and dissipate only 300 mW. This is a 41.3$\times$ reduction in power consumption that could not be provided by a conventional stream processor.

2. Background

2.1 3G base-station processing

Figure 1 shows the architecture of a 32-user base-station with 128 Kbps/user(coded), employing multiuser channel estimation, multiuser detection and Viterbi decoding [5]. Multiuser channel estimation refers to the process of jointly estimating the channel parameters for all users at the base-station. Since the received signal has interference from other users, jointly estimating the parameters allows use to obtain the optimal maximum-likelihood estimate of the channel parameters for all users. However, a maximum likelihood estimate has a significant increase in computational complexity over single-user estimates, but provides a much more reliable channel estimate to the detector. The maximum likelihood solutions also involve matrix inversions, which present difficulties in numerical stability with finite precision computations and in exploiting data parallelism in a simple manner. Hence, we used a conjugate-gradient descent based algorithm that was proposed in [7] that approximates the matrix inversions and replaces the matrix inversion by matrix multiplications, which are simpler to implement and can be computed in finite precision without loss in bit error rate performance. The details of the implemented algorithm are presented in [7]. The computations involve matrix-matrix multiplications of the order of users (U) and the spreading gain (N).

Multi-user detection refers to the joint detection of all the users at the base-station. Since all the wireless users interfere with each other at the base-station, interference cancellation techniques are used to provide reliable detection of the transmitted bits of all users. The detection rate directly impacts the real-time performance. We choose a parallel interference cancellation based detection algorithm [5] for implementation which has a bit-streaming and parallel structure using only adders and multipliers. The computations involve matrix-vector multiplications of the order of the number of active users (U) in the base-station.

Viterbi decoding is used to remove the error control coding done at the transmitter. Viterbi decoding typically involves a trellis [8] with two phases of computation: an add-compare-select phase and a traceback phase. The add-compare-select phase in Viterbi goes through the trellis to compute the most likely path with the least error and has data parallelism proportional to the strength (constraint length) of the code (K). Next, the traceback phase traces the trellis backwards and recovers the transmitted information bits. This traceback is inherently sequential and involves dynamic decisions and pointer-based chasing. With large constraint lengths, the computational complexity of Viterbi decoding increases exponentially and becomes the critical bottleneck, especially as multiple users need to be decoded simultaneously in real-time. The complexity of decoding is proportional to the number of users (U), the constraint length (K) and the coding rate (R). This is the main reason for Viterbi accelerators in the C55x DSP and co-processors in the C6416 DSP from Texas Instruments as the DSP cores are unable to handle the needed computations in real-time. In order to exploit data parallelism for the sequential Viterbi traceback, we use a register-exchange based-scheme [8], which provides a forward traceback scheme with data parallelism that can be exploited in a parallel architecture.

2.2. Wireless evolution

Figure 2 shows the increase in the physical layer operation count from millions of operations per second (MOPs) to billions (giga) of operations per second (GOPs) as a 32-user base-station goes from a second generation (2G) voice only system supporting 16 Kbps/user (coded data rate), to a high data rate third generation (3G) multimedia-oriented system supporting 128 Kbps/user. This two to three orders-of-magnitude increase in complexity is due to several reasons. First, the increase in data rate directly increases the real-time processing requirements at the wireless receiver. Secondly, there is an increase in the computational complexity of the physical layer algorithms in the receiver discussed in the previous section. These advanced algorithms with sophisticated signal processing provide lower bit error rates, resulting in better reception and making more efficient use of the expensive wireless spectrum, enabling higher capacity systems (higher bits/sec/Hz). Note that the numbers shown are for the basic processing operations that are architecture-independent (additions and multiplications) and represent the operations for major compute-intensive parts of the physical layer processing at the base-station. Programmable architectures will have additional overhead to map them onto their instruction set (such as load, store, loop increments, ...) and also need to account for dependencies, register pressure and latencies of the functional units and memory stalls.

Even a voice-oriented 2G base-station design typically requires more than 18 DSPs to meet real-time physical layer processing requirements [1] and extending such DSPs to 3G systems may require greater than a DSP per user [9]. Hence, co-processor designs are currently being proposed to solve this problem. The recently announced 3G TCI1$\times$ UMTS chipset from Texas Instruments for cellular base-stations, contains application-specific hardware for operations such as chip de-spreading and Viterbi/Turbo decoding (DSP co-processors) [10]. Less than 10% of the actual physical layer processing is being done on fully programmable hardware (DSP), which limits the software re-use of the architecture.

2.3. Stream Processors

Current single processor DSP-based architectures do not have sufficient functional units in order to meet real-time requirements for these algorithms. Extending the DSP architectures to add more functional units is not a feasible solution as the register file size and port interconnections rapidly begin to dominate the architecture area [11]. Special-purpose processors for wireless communications perform well because of the abundant parallelism and regular communication patterns within physical layer processing. These processors efficiently exploit these characteristics to keep thousands of arithmetic units busy without requiring many expensive global communication and storage resources. The bulk of the parallelism in wireless physical layer processing can be exploited as data parallelism, as identical operations are performed repeatedly on incoming data elements.

A stream processor can also efficiently exploit this kind of data parallelism, as it processes individual elements from *streams* of data in parallel. Streams are stored in a stream register file, which can efficiently transfer data to and from a set of local register files between major computations. Local register files (LRFs), co-located with the arithmetic units inside the clusters, directly feed those units with their operands. Truly global data, data that is persistent throughout the application, is stored off-chip only when necessary. These three explicit levels of storage form an efficient communication structure to keep hundreds of arithmetic units efficiently fed with data. The Imagine stream processor developed at Stanford is the first implementation of such a stream processor [4].

Figure 3 shows the architecture of a stream processor, with C arithmetic clusters. Operations in a stream processor all consume and/or produce streams which are stored in the centrally located stream register file (SRF). The two major stream instructions are memory transfers and kernel operations. A stream memory transfer either loads an entire stream into the SRF from external memory or stores an entire stream from to the SRF to external memory. Multiple stream memory transfers can occur simultaneously, as hardware resources allow. A kernel operation performs a computation on a set of input streams to produce a set of output streams. Kernel operations are performed within a data parallel array of arithmetic clusters. Each cluster performs the same sequence of operations on independent stream elements. The stream buffers (SBs) allow the single port into the SRF array (limited for area/power/delay reasons) to be time-multiplexed among all the interfaces to the SRF, making it appear that there are many logical ports into the array. The stream buffers (SBs) also act as prefetch buffers and prefetch the data for kernel operations. Both the SRF and stream buffers are banked to match the number of clusters. Hence, kernels that need to access data in other SRF banks need to use the inter-cluster communication network for communicating data between the clusters.

3. Reconfigurable Stream Processors

Stream processors exploit data parallelism available in applications and all clusters execute the same instruction on different data sets in a SIMD manner. In order to use the lowest voltage and frequency (power efficiency),the number of clusters used in a stream processor should be designed depending on the data parallelism available in the application. Table 1 shows the

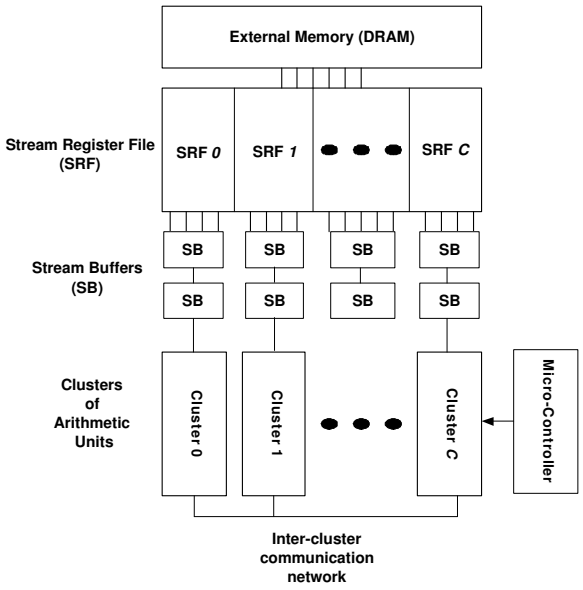

Figure 3. *A Traditional Stream Processor*

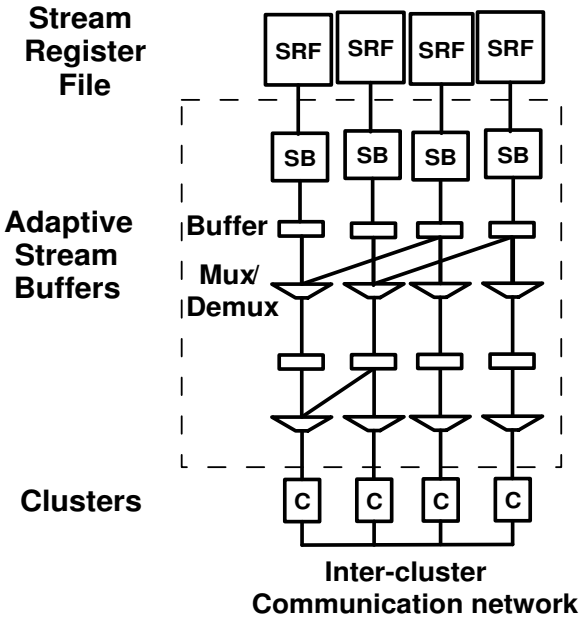

Figure 4. *A Reconfigurable Stream Processor*

available data parallelism for the base-station with variation in the number of users (U) and the decoding constraint length (K). While it is possible to vary other system parameters such as the coding rate (R) and the spreading gain (N) in the table, we decided to keep them constant in order to fix the target data rate to 128 Kbps/user as changing these two parameters affects the target data rate as well in wireless standards [3]. From the table, we can observe that the available data parallelism reduces as we go from the full capacity base-station case (32,9) to lower capacity systems. Based on the data parallelism, if we choose a 32-cluster architecture as the worst case architecture for evaluation, we see that as we go down from case (32,9) to other cases, none of the other workloads meet the minimum data parallelism required to keep all the clusters busy. Hence, there needs to be reconfiguration support provided in stream processors to turn off unused clusters and allow clusters to dynamically match the available data parallelism in the workload.

From Figure 3, we can observe that the input to the clusters arrive directly from the stream buffers, which are banked to match the number of clusters. Thus, if the data parallelism decreases below the number of clusters, the data for the stream will lie in the wrong bank and hence, cannot be accessed directly by the corresponding cluster. Therefore, to successful deactivate clusters in a stream processor, stream data must be rerouted so that active clusters receive the appropriate data from other SRF banks as well. Hence, an interconnection network between the stream buffers and the clusters is needed in order to adapt the clusters to the data parallelism. A fully connected network allowing data to go to any cluster would be extremely expensive in terms of area, power and latency. In fact, stream processors already have a inter-cluster communication network that can be used to route data to any cluster. The intercluster communication network is not only a significant part of cluster power dissipation, but have large latency and area requirements as well.

To investigate better networks, we make use of the fact that it is not necessary to arbitrarily turn off any cluster since all clusters are identical. Hence, we only turn off only those clusters whose cluster identification number is greater than the data parallelism in the algorithms. We further simplify the interconnection network by making the clusters turn off only in powers of two, since most parallel algorithm workloads generally work on datasets in powers of two.

The reconfigurable stream processor is shown in Figure 4. The reconfigurable stream processor allows the ability to turn the clusters on and off, depending on the available data parallelism using an interconnection network within the stream buffers. This interconnection network allows the stream buffers to become adaptive to the workload. In the absence of any reconfiguration needed such as in case (32,9) of Table 1, the interconnection network acts as an extended stream buffer, providing the ability to prefetch more data while behaving as a reconfiguration support when the data parallelism reduces below the number of clusters.

Figure 5 shows how the adaptive stream buffers would operate if reconfiguration is needed. The switches in the mux/demux network are set to control the flow of the data to the clusters. There are $\log_2(C)$ stages required in the stream buffer if complete reconfiguration to a single cluster is desired, where C is the number of clusters. The reconfiguration network allows the stream processor to stripe and access data across all the SRF banks and provides high memory efficiency as well instead of limiting

Workload (U,K)	Estimation f(U,N)	Detection f(U,N)	Decoding f(U,K,R)
(4,7)	32	4	16
(4,9)	32	4	64
(8,7)	32	8	16
(8,9)	32	8	64
(16,7)	32	16	16
(16,9)	32	16	64
(32,7)	32	32	16
(32,9)	32	32	64

Table 1. *Available Data Parallelism in Wireless Communication Workloads (U = Users, K = constraint length, N = spreading gain (fixed at 32), R = coding rate(fixed at rate 1/2))*

Input Data from SB : **ABCDEFGH**			
Time	**Stage 1**	**Stage 2**	**Cluster**
1	ABCD	-	-
2	ABCD	AB	-
3	ABCD	AB	A
4	ABCD	CD	B
5	EFGH	CD	C
6	EFGH	EF	D
7	EFGH	EF	E
:	:	:	:

Table 2. *Example for 4:1 reconfiguration case (c) of Figure 5. Y-axis represents time. Data enters cluster 1 sequentially after an additional latency of 2 cycles*

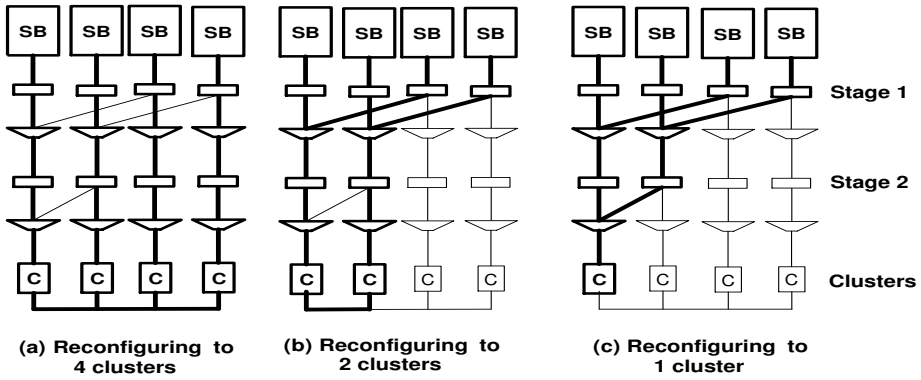

Figure 5. *Reorganizing streams with an adaptive stream buffer*

the data storage to that of the parallelism present in the data stream and zero-padding unused SRF banks. The adaptive stream buffers also provide higher SRF bandwidth to applications with insufficient data parallelism since all the SRF banks can be accessed and utilized even in cases with insufficient data parallelism.

The example in Table 2 shows a 4-cluster stream processor reconfiguring to 1 cluster. We can see that each stage of the multiplexer network holds the data until it is completely read into the clusters. This is done using counters for each stream buffer stage (not shown for clarity). Also, stalls need to be handled by the multiplexer network. The adaptive stream buffer network adds a latency of $\log_2(C)$ to the data entering the clusters. However, the adaptive stream buffers can prefetch data even if the clusters are stalled, allowing it to have the potential to hide latencies or even improve performance if clusters have significant stalls.

The multipexer/demultiplexer network shown in Figure 4 is per data stream. Each input stream is configured as a multiplexer and each output stream is configured as a demultiplexer during execution of a kernel. The cluster reconfiguration is done dynamically by providing support in the instruction set of the host processor for setting the number of active clusters during the execution of a kernel. By default, all the clusters are turned off when kernels are inactive, thereby providing power savings during memory stalls and system initialization. Another advantage of providing reconfiguration using the multiplexer network is that users do not have to modify their kernel code to account for this reconfiguration support. A 'setClusters(*int* clusters)' is added in the stream code to set the number of clusters required for the kernel execution. This instruction is scheduled to execute just before the kernel executes and gets reset at the end of the kernel to zero so that the clusters can be turned off for the maximum time period possible.

The current implementation of the multiplexer network adds a latency of $\log_2(C) + 2$ cycle for the data to flow through the multiplexer network. Hence, all the data entering the clusters are delayed by 7 cycles for a 32-cluster stream processor. However, the multiplexer network behaves as an extended stream buffer and allows data prefetching to occur in the multiplexer network, allowing the ability to absorb the latency. Also, it is possible that the kernel does not read the data during the first 7 cycles (or the kernel can be re-scheduled by the compiler such that the first read occurs after 7 cycles). However, the current implementation stalls the clusters for 7 cycles for all kernels. In spite of this, we can see a reduction in the micro-controller

Kernel	Calls	Base		New	
		busy	stalls	busy	stalls
Update channel matrix	1	223	544	223	533
Matrix multiplication	1	12104	4934	11464	5324
Iterate for new estimates	1	272	996	206	1057
Matrix mult. for L	1	6537	67	6218	22
Matrix mult. for C	1	6826	2075	6506	2052
Matrix transpose	5	1075	1425	1070	1380
Matched Filter	67	18492	2479	18492	3752
Interference Cancellation	197	35854	13396	35854	17139
Pack data	1	289	0	289	7
Repack data	6	444	24	444	54
Viterbi Init.	32	4736	0	4736	224
Add Compare Select	1024	114688	0	114688	7168
Extract Decoded bits	32	5952	0	5792	224
Kernel Time		223432	25940	224918	38936
Memory Time		-	25261	-	34645
Total Time		258693		279563	
Real-time frequency		1.03 GHz		1.12 GHz	

Table 3. *Worst case reconfigurable stream processor performance*

stalls for some of the kernels. Table 3 compares the reconfigurable stream processor with the base stream processor for the full capacity base-station case (32,9) that does not require any reconfiguration, thereby allowing us to evaluate the worst case performance of the dynamically reconfigurable processor. From the table, we can observe that for kernels such as 'matrix multiplication for L', there was a reduction in the stall time due to the adaptive stream buffers, although the net result was a slight degradation in performance. The current implementation for the reconfigurable stream processor needs to run at 1.12 GHz to meet real-time constraints as opposed to 1.03 GHz for a traditional stream processor when no reconfiguration is required. However, by turning off the clusters when memory stalls are present, the reconfigurable stream processors can still more power-efficient even in the no reconfiguration case, and this is will be shown in the next section.

The power model for stream processors is based on [12] with additions to account for the adaptive stream buffers. The parameters used in power analysis in this paper are as shown in Table 4 and Table 5. All energy values shown are with respect to E_w. Based on the models in [12], with modifications to account for fixed point ALUs, the length of a wire track is 0.455 micron, a cluster is 1400 wire tracks wide and it takes 0.42 pJ to propagate a signal across a single wire track. The modifications are based on a low power version of stream processors for embedded systems (See (LP) in Chapter 6 in [13]). The power consumption of the mux network is wire length capacitance dominant to a first order approximation [12]. Assuming a linear layout of the clusters (worst case), the total wire length of the mux network for 1 bit is 341 units, where the unit is the distance between 2 clusters. For 8 streams with 32 bits of wire each, the mux network uses a total wire of length approximately $341 * 1400 * 8 * 32 * 0.455 = 55$ meters with a power consumption of 5 μJ [12]. The clusters can be re-ordered according to bit-reversal of their locations and this can be shown to minimize the wire length from 341 units per bit to 80 units per bit, (from O(n^2) to O($n * \log(n)$)), where n is the number of clusters. The power consumption of this network is less than 1% of a single multiplier and this has a negligible effect on the power consumption of the stream processor design. This power has been accounted for in the simulation numbers presented in the next section.

4. Results

Figure 6 shows the clock frequency needed by the reconfigurable stream processor to meet real-time requirements of 128 Kbps/user. We can see that as the workload decreases, the percentage of cluster busy time and microcontroller stalls decrease. However, the memory stall time does not seem to decrease with the workload. This is because as the workload decreases from (32,9) to the (4,7) case, some of the memory stalls that were hidden during kernel execution suddenly became visible due to the corresponding decrease in the kernel execution time. The reconfigurable stream processor needs to run at 1.12 GHz to meet real-time for the full capacity workload and can run at 345.1 MHz when the workload decreases to the (4,7) case.

Figure 7 shows the corresponding cluster utilization variation with the workload and the cluster index. We can see that in the full capacity case (32,9), all clusters are equally utilized at 87%. The clusters are idle and are turned off 13% of the time due to memory stalls. However, as the workload decreases, the reconfigurable stream processor turns off unutilized clusters to

Description	Value
ALU operation energy	632578
LRF access energy	79365
Energy of SB access/bit	155
SRAM access energy/bit	8.7
SP access energy	1603319
Norm. wire energy/track(E_w)	1

Table 4. *Power consumption table (normalized to E_w units)*

Parameter	Value
No. of clusters	32
Adders/cluster	3
Multipliers/cluster	3
Stream block size	32 words
Stream Buffer size	256 KB
DRAM frequency	$f_{CLK}/8$
Adder latency	2 cycles
Multiplier latency	4 cycles

Table 5. *Key simulation parameters*

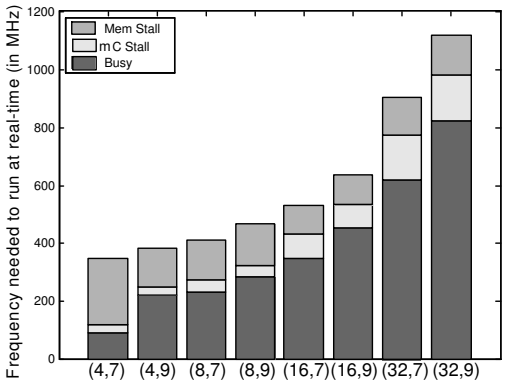

Figure 6. *Clock Frequency needed to meet real-time requirements with varying workload*

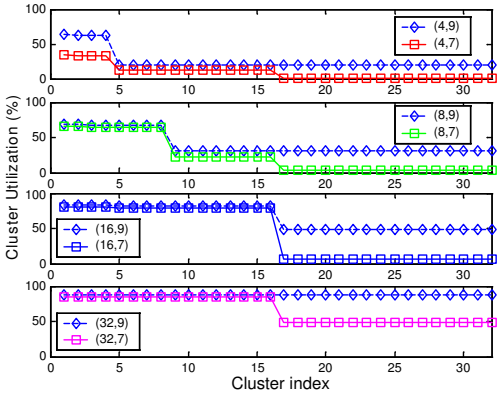

Figure 7. *Cluster utilization variation with cluster index and with workload*

lower their utilization factor and save power. For example, we can see in case (4,9) that only the first 4 clusters are being used at 63% utilization, while the remaining 28 clusters are being used at 20% utilization. The adaptive stream buffer provides the needed data alignment to collect the data from all the 32 SRF banks and stream buffers into only those clusters that are active. Thus, by turning off unused clusters during periods of memory stalls and insufficient data parallelism, reconfigurable stream processors are able to provide power-efficient wireless base-stations.

However, it is not practical to run the clock of the reconfigurable stream processor at just the right frequency to meet real-time. There are only a few possible frequency levels in programmable processors and these are standardized due to interface with external DRAM clock frequencies. Hence, the reconfigurable stream processor needs to be over-clocked to work at these fixed frequencies. However, the clusters can be turned off during spare cycles now available as well. In this paper, we assume frequencies and voltages used in the latest TM5800 Transmeta Crusoe chip [14], with an extension to 1.2 GHz case at 1.4 V.

Power saving is achieved in the reconfigurable stream processor due to turning off clusters during over-clocking (the idle time due to mismatch between the frequency needed and the actual frequency used), during memory stalls and during kernels having clusters with insufficient data parallelism. This is shown in Table 6. In order to evaluate the benefits of the adaptive stream buffers, the base case comparison is assumed to be a stream processor that already supports dynamic voltage and frequency scaling. We can see from the table that the adaptive stream buffers yields savings in power even in the no reconfiguration case (32,9) of a full capacity base-station due to turning off clusters during memory stalls and over-clocking. We can see that the adaptive stream buffers yield an additional 15-85% power savings over that provided by simple frequency and voltage scaling in the architecture.

5. Discussion and Conclusions

In addition to turning off clusters, functional units within the cluster can be turned off as well during the execution of different kernels. This is because the functional unit utilization for the kernels can be approximately pre-determined during compile time due to the static nature of the workload (a reasonable approximation, as micro-controller stalls are ignored). This can be used to turn off adders and multipliers to save static power dissipation. Note that the dynamic power dissipation will

Workload	Freq (MHz)		Voltage	Power Savings (W)			Power (W)		Savings
	needed	used	(V)	clocking	Memory	Clusters	New	Base	
(4,7)	345.09	433	0.875	0.325	1.05	0.366	0.30	2.05	85.14 %
(4,9)	380.69	433	0.875	0.193	0.56	0.604	0.69	2.05	66.41 %
(8,7)	408.89	433	0.875	0.089	0.54	0.649	0.77	2.05	62.44 %
(8,9)	463.29	533	0.950	0.304	0.71	0.643	1.33	2.98	55.46 %
(16,7)	528.41	533	0.950	0.020	0.44	0.808	1.71	2.98	42.54 %
(16,9)	637.21	667	1.050	0.156	0.58	0.603	3.21	4.55	29.46 %
(32,7)	902.89	1000	1.300	0.792	1.18	1.375	7.11	10.46	32.03 %
(32,9)	1118.25	1200	1.400	0.774	1.41	0.000	12.38	14.56	14.98 %
Estimated Cluster Power Consumption									78 %
Estimated SRF Power Consumption									11.5 %
Estimated Microcontroller Power Consumption									10.5 %
Estimated Chip Area									45.7 mm^2

Table 6. *Power Savings*

still be the same as the number of operations will not change. We have observed that this can yield up to an additional 5-10% power savings.

While turning off clusters during memory stalls, it is important to take care that data inside the clusters are flushed and no variables persist internally in the cluster scratchpad for utilization in the next kernel. In our case, only decoding uses persistent data and does not have any memory stalls. An instruction can also be added to the compiler to check whether persistent data is declared in the kernels before turning off clusters before the next kernel.

In conclusion, we have demonstrated a real-time advanced 32-user base-station implementation at 128 Kbps/user using reconfigurable stream processors that provide high performance (GOPs) and are power-efficient (300 mW - 12.38 W). An adaptive stream buffer network is introduced that allows clusters to be turned off dynamically for power efficiency and is shown to give 15-85% power savings over a stream processor with just voltage and frequency scaling. This adaptive stream buffer feature can be used for power efficiency in other stream and vector processor applications as well when the data parallelism falls below the number of processors existing in the architecture.

References

[1] N. Mera, "The wireless challenge," Telephony Online, http://telephonyonline.com/ar/telecom_wireless_challenge, December 9 1996.

[2] Paul R. La Monica, "Wireless gets blacked out too - North America Power Blackout," CNN Money News report: http://money.cnn.com/2003/08/15/technology/landlines/index.htm, August 16 2003.

[3] "Third Generation Partnership Project," http:// www.3gpp.org.

[4] U. J. Kapasi, S. Rixner, W. J. Dally, B. Khailany, J. H. Ahn, P. Mattson, and J. D. Owens, "Programmable stream processors," *IEEE Computer*, vol. 36, no. 8, pp. 54–62, August 2003.

[5] S. Rajagopal, S. Rixner, and J. R. Cavallaro, "A programmable baseband processor design for software defined radios," in *IEEE International Midwest Symposium on Circuits and Systems*, Tulsa, OK, August 2002.

[6] M. Grollo, W. A. Cornelius, M. J Bangay, and E. E. Essex, "Radiofrequency radiation emissions from mobile telephone base station communication towers," in *IEEE Engineering in Medicine and Biology Society: Biomedicial Research in the 3rd Millennium*, Victoria,Australia, February 1999.

[7] S. Rajagopal, S. Bhashyam, J. R. Cavallaro, and B. Aazhang, "Real-time algorithms and architectures for multiuser channel estimation and detection in wireless base-station receivers," *IEEE Transactions on Wireless Communications*, vol. 1, no. 3, pp. 468–479, July 2002.

[8] S. B. Wicker, *Error Control Systems for Digital Communication and Storage*, Prentice Hall, 1 edition, 1995.

[9] "Implementation of a WCDMA Rake receiver on a TMS320C62x DSP device," Tech. Rep. SPRA680, Texas Instruments, July 2000.

[10] "Channel card design for 3G infrastructure equipment," Tech. Rep. SPRY048, Texas Instruments, 2003.

[11] S. Rixner, W. Dally, B. Khailany, P. Mattson, U. Kapasi, and J. Owens, "Register organization for media processing," in *6th International Symposium on High-Performance Computer Architecture*, Toulouse, France, January 2000, pp. 375–386.

[12] B. Khailany, W. J. Dally, S. Rixner, U. J. Kapasi, J. D. Owens, and B. Towles, "Exploring the VLSI scalability of stream processors," in *International Conference on High Performance Computer Architecture (HPCA-2003)*, Anaheim, CA, February 2003.

[13] B. Khailany, *The VLSI implementation and evaluation of area- and energy-efficient streaming media processors*, Ph.D. thesis, Stanford University, Palo Alto, CA, June 2003.

[14] Transmeta, "Crusoe Processor Product Brief: Model TM5800," http://www.transmeta.com/pdf/specifications/TM5800_productbrief_030206.pdf.